# Software Verification & Validation

Comprehensive Course Notes

---

Subject: Software Verification and Validation | Credit Hours: 3 | Prerequisite: Intro to Software Engineering

---

## 1. Introduction to Software Quality Assurance

*Software Quality Assurance (SQA) is a set of planned and systematic activities applied throughout the software development lifecycle to ensure that software products and processes conform to defined requirements and standards.*

### Core Concepts

- **Quality:** The degree to which software meets specified requirements and satisfies customer expectations.
- **Assurance:** Confidence established through evidence that quality requirements will be fulfilled.
- **Process vs Product quality:** SQA covers both the process used to build software and the product itself.

### Goals of SQA

- Prevent defects from entering the software product.
- Detect defects early in the SDLC when they are cheapest to fix.
- Ensure adherence to standards, procedures, and guidelines.
- Provide management with visibility into the development process.
- Continuously improve development processes.

### SQA Activities

- **Standards establishment:** Define coding standards, design standards, documentation templates.
- **Reviews and audits:** Peer reviews, walkthroughs, formal inspections of work products.
- **Testing coordination:** Ensuring comprehensive test coverage across all levels.
- **Configuration management:** Controlling changes to software artifacts.
- **Metrics collection:** Gathering data on defect rates, process efficiency, and product quality.

### SQA vs Software Testing

| Dimension | SQA | Software Testing |
|---|---|---|
| Scope | Entire SDLC process | Product/code execution phase |
| Focus | Prevention of defects | Detection of defects |
| Who | SQA team + all stakeholders | QA testers / developers |
| Output | Process improvement reports | Bug reports, test results |

### SQA Framework Elements

- **Pre-project planning:** Define quality objectives before development begins.
- **Standards and procedures:** Document accepted practices for each development phase.
- **Reviews:** Structured evaluations of intermediate work products.
- **Testing:** Systematic execution of software to find defects.
- **Defect tracking:** Recording, classifying, and following up on defects.
- **Process improvement:** Using data to refine future development practices.

---

# 2. The Quality Challenge

*The Quality Challenge refers to the inherent difficulty in consistently producing high-quality software due to the complexity of software systems, human error, evolving requirements, and the intangible nature of software.*

### Why Quality Is Difficult to Achieve

- **Complexity:** Modern software contains millions of lines of code with intricate interdependencies.
- **Invisibility:** Software has no physical form — defects cannot be seen or touched.
- **Changeability:** Requirements and environments change continuously during and after development.
- **Human error:** Software is designed and written by people who make mistakes.
- **Scale:** Large teams, distributed development, and long timelines increase coordination errors.

### Common Quality Problems

- **Ambiguous requirements:** Unclear specifications lead to incorrect implementations.
- **Poor design decisions:** Architectural flaws that propagate throughout the codebase.
- **Inadequate testing:** Insufficient test coverage leaves defects undetected.
- **Time pressure:** Deadlines lead to skipped reviews and incomplete testing.
- **Lack of standards:** Inconsistent practices across teams produce uneven quality.

### The Cost of Poor Quality

- **Internal failure costs:** Costs of fixing defects found before delivery (rework, retesting).
- **External failure costs:** Costs of defects found after delivery (support, recalls, liability).
- **Appraisal costs:** Cost of measuring quality (reviews, testing, audits).
- **Prevention costs:** Investment in processes to avoid defects (training, standards, tools).

---

**Rule of Ten:** The cost to fix a defect multiplies by roughly 10x for each phase it passes through undetected (requirements → design → code → test → production).

---

### Responses to the Quality Challenge

- Establish formal SQA programs with defined roles and responsibilities.
- Apply quality standards (ISO 9001, IEEE standards, CMMI).
- Use metric-driven quality management to objectively assess quality.
- Invest in prevention through training, tool support, and process improvement.
- Conduct independent quality audits and reviews.

---

# 3. Quality Control vs. Quality Assurance

*Quality Control (QC) is a reactive, product-focused activity that detects defects in completed work products, while Quality Assurance (QA) is a proactive, process-focused activity that prevents defects by improving development processes.*

### Key Differences

| Dimension | Quality Control (QC) | Quality Assurance (QA) |
|---|---|---|
| Nature | Reactive | Proactive |
| Focus | Product / output | Process / methodology |
| Goal | Detect and fix defects | Prevent defects from occurring |
| Timing | After work is produced | Throughout the entire SDLC |
| Who performs | Testers, reviewers | SQA team, process engineers |
| Techniques | Testing, inspection, review | Audits, standards, process reviews |
| Output | Defect reports | Process improvement plans |

### Quality Control Activities

- Code reviews and walkthroughs — reviewing written code for defects.
- Testing (unit, integration, system, acceptance).
- Inspection of design documents and requirements specifications.
- Static analysis using automated tools.
- Defect logging and tracking until closure.

### Quality Assurance Activities

- Defining and enforcing development standards and procedures.
- Conducting process audits to verify compliance.
- Training developers in quality techniques.
- Managing configuration and change control processes.
- Collecting and analyzing process metrics for improvement.

### Relationship Between QA and QC

- QA sets up the framework; QC operates within it.
- QA ensures that QC processes are adequate and followed.
- Both are necessary — QA alone cannot guarantee quality without QC execution.
- In mature organizations, QA reduces the volume of defects that QC must handle.

---

## 4. Quality Assurance in Software Projects (Phases)

*SQA is applied at every phase of the Software Development Lifecycle (SDLC) — not only at the end — to ensure quality is built into the product from the start.*

### SQA in Requirements Phase

- Review requirements specifications for completeness, consistency, and clarity.
- Verify that requirements are testable.

- Identify ambiguous or conflicting requirements early.
- Establish traceability matrix linking requirements to test cases.

### SQA in Design Phase

- Review architectural and detailed design documents.
- Check design against requirements (design verification).
- Evaluate design for adherence to standards (modularity, coupling, cohesion).
- Conduct formal design inspections to detect structural defects.

### SQA in Coding Phase

- Enforce coding standards and guidelines.
- Conduct peer code reviews and pair programming.
- Apply static analysis tools (linters, code analyzers).
- Track code coverage metrics during unit testing.

### SQA in Testing Phase

- Verify test plans and test cases for completeness.
- Monitor test execution and defect discovery rates.
- Ensure all requirement areas are covered by test cases.
- Review and validate test results.

### SQA in Maintenance Phase

- Review change requests for impact on existing functionality.
- Ensure regression testing is performed after each change.
- Audit change control procedures for compliance.
- Track and analyze field defects for process improvement.

### Phase-by-Phase SQA Summary

| SDLC Phase | SQA Activity | Primary Output |
|---|---|---|
| Requirements | Requirements review | Defect-free SRS |
| Design | Design inspection | Approved design document |
| Coding | Code review + static analysis | Reviewed, standards-compliant code |
| Testing | Test plan review + monitoring | Test completion report |
| Maintenance | Change review + regression audit | Controlled, regression-tested release |

---

## 5. Principles and Practices of SQA

*SQA principles are the fundamental rules that guide how quality assurance is applied, while practices are the concrete techniques and activities used to implement those principles.*

### Core SQA Principles

- **Prevention over detection:** Address quality at the source rather than finding and fixing defects later.
- **Process discipline:** Follow defined, documented, and consistently applied processes.

- **Independence:** SQA personnel should be organizationally independent from development teams.
- **Measurement:** Quality must be measured objectively using quantitative metrics.
- **Continuous improvement:** Use collected data to iteratively improve processes.
- **Visibility:** Quality status and issues must be visible to management.
- **Early involvement:** SQA must begin at the project inception, not the testing phase.

### Key SQA Practices

- **Standards adoption:** Use industry standards (ISO, IEEE) as baselines for process definition.
- **Formal inspections:** Structured, role-based examination of work products against defined checklists.
- **Metrics programs:** Systematic collection of defect data, effort data, and process performance data.
- **Configuration management:** Version control and change management of all project artifacts.
- **Risk management:** Identifying and mitigating quality risks before they cause failures.
- **Training programs:** Ensuring all team members are equipped with SQA knowledge.

### SQA in Agile Contexts

- SQA principles still apply but are integrated into short iterations (sprints).
- Continuous integration and automated testing replace periodic formal reviews.
- Quality ownership is shared by the entire team, not a separate SQA department.
- Definition of Done (DoD) acts as a built-in quality gate per sprint.

---

## 6. Quality Management

*Quality Management is the overall discipline of planning, assuring, controlling, and improving quality across all organizational processes that produce software products.*

### Four Pillars of Quality Management

- **Quality Planning:** Setting quality objectives, determining standards, and identifying which quality processes apply.
- **Quality Assurance:** Systematic activities providing confidence that quality requirements will be fulfilled.
- **Quality Control:** Operational techniques to monitor and eliminate causes of unsatisfactory performance.
- **Quality Improvement:** Actions taken to increase effectiveness and efficiency of processes.

### Quality Management Models

- **CMMI (Capability Maturity Model Integration):** Five levels of process maturity — Initial, Managed, Defined, Quantitatively Managed, Optimizing.
  - Level 1 (Initial): Ad hoc processes, unpredictable.
  - Level 2 (Managed): Project-level process control established.
  - Level 3 (Defined): Organizational standards defined and deployed.
  - Level 4 (Quantitatively Managed): Process performance measured statistically.
  - Level 5 (Optimizing): Continuous process improvement using data.
- **ISO 9001:** International standard for quality management systems based on customer focus, leadership, process approach, and continual improvement.
- **Six Sigma:** Data-driven methodology targeting near-zero defect rates (3.4 defects per million opportunities).

### Quality Management Roles

- **Quality Manager:** Oversees SQA program, reports to senior management.

- **SQA Engineers:** Perform audits, reviews, metrics collection.
- **Process Engineers:** Define and improve development processes.
- **Project Managers:** Responsible for implementing quality plans at project level.

### Quality Metrics in Management

- **Defect density:** Number of defects per KLOC (thousand lines of code).
- **Defect removal efficiency (DRE):** Percentage of defects removed before delivery.
- **Customer satisfaction index:** Post-delivery measurement of user satisfaction.
- **Test coverage:** Percentage of code or requirements covered by tests.
- **Mean Time Between Failures (MTBF):** Average time a system runs without failure.

---

## 7. Quality Assurance and Standards

*Quality standards are documented agreements defining specifications, procedures, or guidelines to ensure that materials, products, processes, and services are fit for their purpose.*

### Major Software Quality Standards

- **ISO 9001:2015** — Generic quality management system standard applicable to software organizations.
  - Requires documented processes, management commitment, and customer focus.
  - Certification involves third-party audits.
- **ISO/IEC 25010** (formerly ISO 9126) — Defines software product quality model with 8 characteristics.
  - Functional suitability, Performance efficiency, Compatibility, Usability.
  - Reliability, Security, Maintainability, Portability.
- **IEEE 730** — Standard for Software Quality Assurance Plans (SQAP).
- **IEEE 829** — Standard for software test documentation.
- **DO-178C** — Software considerations in airborne systems and equipment certification.
- **IEC 61508** — Functional safety of electrical/electronic/programmable electronic safety-related systems.

### Using Standards in SQA

- Standards provide a baseline — teams adapt them to their specific context.
- Compliance with standards must be verified through audits, not assumed.
- Standards evolve — organizations must track and adopt updated versions.
- Contractual requirements may mandate adherence to specific standards.

### Standards vs. Guidelines vs. Procedures

| Term | Definition | Mandatory? |
|---|---|---|
| Standard | Formal, agreed-upon specification | Yes (when adopted) |
| Guideline | Recommended practice, not binding | No |
| Procedure | Step-by-step instructions for a specific activity | Yes (when defined) |

---

## 8. Quality Planning and Quality Control

*Quality Planning is the process of identifying quality standards relevant to the project and determining how to satisfy them; Quality Control is the operational monitoring of project results to ensure standards are met.*

### Quality Planning — Steps

- **Step 1 — Define quality objectives:** Specify measurable quality targets (e.g., defect density < 1 per KLOC).
- **Step 2 — Identify standards:** Select applicable standards (IEEE, ISO, domain-specific).
- **Step 3 — Choose QA activities:** Decide which reviews, inspections, and audits will be conducted.
- **Step 4 — Define metrics:** Specify which metrics will be collected and how they will be used.
- **Step 5 — Resource planning:** Assign personnel, tools, and schedule for SQA activities.
- **Step 6 — Document the plan:** Produce the Software Quality Assurance Plan (SQAP).

### Quality Control — Techniques

- **Statistical Process Control (SPC):** Monitoring process outputs using control charts to detect when a process goes out of control.
- **Cause-and-effect (Ishikawa) diagrams:** Identifying root causes of defects.
- **Pareto analysis:** 80/20 rule — identifying the 20% of defect types causing 80% of problems.
- **Control charts:** Plotting process metrics over time to identify trends and variation.
- **Histograms:** Showing distribution of defects by type, phase, or module.

### Quality Control — Process

- **Monitor:** Continuously collect data on actual process performance.
- **Compare:** Compare actual data against quality baselines and standards.
- **Analyze:** Identify deviations, trends, and root causes.
- **Act:** Apply corrective actions to bring process back into control.

### Key Metrics for Quality Control

- Defect detection rate by phase.
- Defect fix rate and backlog.
- Test case pass/fail rates over time.
- Rework effort as percentage of total effort.
- Schedule and budget variance correlated with quality events.

---

## 9. Verification and Validation (V&V)

*Verification asks 'Are we building the product right?' (conformance to specifications), while Validation asks 'Are we building the right product?' (fitness for user needs).*

### Formal Definitions (IEEE 1012)

- **Verification:** The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
- **Validation:** The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

### V&V Comparison

| Aspect | Verification | Validation |
|---|---|---|
| Question | Built it right? | Built the right thing? |
| Reference | Specifications / design docs | User needs / requirements |
| Timing | During each development phase | At end of development |
| Techniques | Reviews, inspections, walkthroughs | Testing, demos, user acceptance |
| Who performs | Internal team / SQA | Client / end users / SQA |

### V&V Activities

- **Requirements V&V:** Check that requirements are complete, consistent, feasible, and testable.
- **Design V&V:** Confirm design correctly implements requirements and follows design standards.
- **Code V&V:** Verify code correctly implements design through reviews and static analysis.
- **Test V&V:** Validate that test results demonstrate system meets user requirements.

### V Model (V-Diagram)

- Left side of V — development phases (requirements → design → coding).
- Right side of V — testing phases (unit test → integration test → system test → acceptance test).
- Each right-side activity validates the corresponding left-side artifact.
- Ensures systematic traceability between development and testing.

### Static vs Dynamic V&V

| Type | Description | Examples |
|---|---|---|
| Static | No code execution — examining artifacts | Reviews, inspections, walkthroughs |
| Dynamic | Code is executed with test inputs | Unit testing, integration testing, system testing |

---

## 10. Planning Verification and Validation

*V&V planning is the process of defining the scope, approach, resources, and schedule for all V&V activities across the entire SDLC, documented in a Software V&V Plan (SVVP).*

### IEEE 1012 — SVVP Contents

- **Purpose:** Define which V&V tasks will be performed and why.
- **Scope:** Identify the software items subject to V&V.
- **References:** List applicable standards, plans, and documents.
- **Definitions:** Clarify terminology used in the plan.
- **V&V overview:** Organization, schedule, resources, responsibilities, tools.
- **Life cycle V&V requirements:** Specific V&V activities per phase.
- **Agency role:** Responsibilities of each organization in V&V.

### Planning Considerations

- **Criticality level:** More critical systems require more rigorous V&V (safety-critical vs non-critical).
- **Independence:** Decide whether V&V is internal or performed by an independent body.
- **Entry/exit criteria:** Define conditions for starting and completing each V&V activity.

- **Resource allocation:** People, tools, environments, test data required.
- **Risk-based prioritization:** Focus V&V effort on highest-risk components.

### V&V Integrity Levels

- IEEE 1012 defines four integrity levels based on system criticality.
  - Level 4 (highest): Safety-critical, human life at risk — maximum V&V rigor.
  - Level 3: Mission-critical, significant financial or operational impact.
  - Level 2: Important system, moderate consequences of failure.
  - Level 1 (lowest): Minimal consequence of failure — minimum V&V.
- Higher integrity levels require more V&V techniques and more thorough documentation.

---

## 11. Critical System Validation

*Critical system validation is the process of rigorously demonstrating that a safety-critical, mission-critical, or security-critical system satisfies its requirements and will not cause unacceptable harm under all operating conditions.*

### Types of Critical Systems

- **Safety-critical:** Failure could result in loss of life or injury (medical devices, aircraft control, nuclear plant software).
- **Mission-critical:** Failure causes significant operational or financial loss (banking systems, military command systems).
- **Business-critical:** Failure disrupts normal business operations significantly.
- **Security-critical:** Failure leads to unauthorized access or data breach.

### Challenges in Critical System Validation

- Exhaustive testing is impossible — infinite input space.
- Rare failure scenarios are difficult to reproduce in test environments.
- Emergent behaviors from system integration are hard to anticipate.
- Regulatory certification requirements add overhead and constraints.

### Critical System Validation Techniques

- **Formal methods:** Mathematical specification and proof of system correctness (model checking, theorem proving).
- **Fault tree analysis (FTA):** Top-down deductive analysis identifying causes of system failure.
- **Failure Mode and Effect Analysis (FMEA):** Bottom-up analysis of each component failure and its effect.
- **Hazard analysis:** Identifying situations that could lead to accidents.
- **Stress and boundary testing:** Validating system behavior under extreme conditions.
- **Independent validation:** Third-party team validates the system against specifications.

### Regulatory Standards for Critical Systems

- **IEC 61508:** Functional safety — defines Safety Integrity Levels (SIL 1–4).
- **DO-178C:** Aviation software — defines Design Assurance Levels (DAL A–E).
- **IEC 62304:** Medical device software — lifecycle requirements.
- **EN 50128:** Railway applications — software for railway control and protection systems.

---

# 12. Reliability Validation

*Reliability validation is the process of determining whether a software system meets its specified reliability requirements, expressed as the probability of failure-free operation for a specified time in a specified environment.*

### Key Reliability Metrics

- **MTBF (Mean Time Between Failures):** Average time a system operates between failures.
- **MTTF (Mean Time To Failure):** Expected time before the first failure (for non-repairable systems).
- **MTTR (Mean Time To Repair):** Average time to restore system after a failure.
- **Availability:** MTBF / (MTBF + MTTR) — fraction of time the system is operational.
- **Failure rate (λ):** Number of failures per unit time; λ = 1 / MTBF.

### Software Reliability Models

- **Jelinski-Moranda model:** Failure rate decreases after each bug is found and fixed.
- **Littlewood-Verrall model:** Assumes debugging may introduce new defects.
- **NHPP (Non-Homogeneous Poisson Process):** Models cumulative failures as Poisson process with varying rate.
- **Musa Execution Time model:** Reliability measured in CPU execution time rather than calendar time.

### Reliability Validation Process

- **Step 1 — Define reliability targets:** Specify MTBF, failure rate, or availability requirements.
- **Step 2 — Operational profile:** Model how the system will actually be used (input frequency distribution).
- **Step 3 — Reliability testing:** Execute tests drawn from operational profile to measure actual failure rate.
- **Step 4 — Statistical analysis:** Apply reliability models to observed failure data.
- **Step 5 — Compare with targets:** Determine if measured reliability meets specified requirements.

### Reliability Growth Testing

- System is tested, failures are found and fixed, and reliability is measured over time.
- Reliability should grow (failure rate decrease) as defects are removed.
- Growth is plotted and projected to determine when reliability targets will be met.
- Used to make release decisions based on demonstrated reliability, not schedule.

---

# 13. Safety Assurance

*Safety assurance is the systematic set of activities that provides confidence that a safety-critical software system will not cause unacceptable harm to people, property, or the environment.*

### Core Concepts

- **Hazard:** A condition or event that could lead to an accident.
- **Risk:** The combination of the probability of a hazard occurring and the severity of its consequences.
- **Safety requirement:** A constraint derived from hazard analysis that the system must satisfy.
- **Safety integrity level (SIL):** A discrete level (1–4) specifying the required risk reduction.

### Safety Assurance Process

- **Hazard identification:** Use HAZOP, FTA, FMEA to identify all potential hazards.
- **Risk assessment:** Estimate probability and severity for each hazard.
- **Risk mitigation:** Define requirements and design features to reduce risk to acceptable levels.
- **Safety case:** Documented structured argument with evidence that the system is safe.
- **Independent safety assessment:** Third-party review of safety case and evidence.

### Safety Analysis Techniques

| Technique | Approach | Use |
|---|---|---|
| FTA (Fault Tree Analysis) | Top-down, deductive | Find causes of a specific failure |
| FMEA | Bottom-up, inductive | Find effects of each component failure |
| HAZOP | Guided word analysis | Identify hazards in design/process |
| Event Tree Analysis | Forward, inductive | Trace consequences of initiating event |

### Safety Requirements Properties

- Shall not permit the system to enter a known unsafe state.
- Shall ensure safe shutdown or graceful degradation on failure.
- Shall enforce safe limits on all physical parameters (speed, pressure, dose).
- Shall be derived directly from documented hazard analysis results.

---

## 14. Security Assessment

*Security assessment is the systematic process of identifying, evaluating, and mitigating security vulnerabilities and threats in a software system to protect against unauthorized access, data breaches, and malicious attacks.*

### Security Assessment Types

- **Vulnerability assessment:** Scanning for known vulnerabilities in the system (automated tools — NIST NVD, CVE databases).
- **Penetration testing (pen testing):** Simulating real attacks to discover exploitable vulnerabilities.
- **Threat modeling:** Systematically identifying potential attackers, attack vectors, and assets at risk.
- **Security code review:** Manual or automated examination of source code for security flaws.
- **Security audit:** Formal examination of security policies, controls, and compliance.

### Threat Modeling — STRIDE

- **S — Spoofing:** Impersonating a user or system component.
- **T — Tampering:** Unauthorized modification of data.
- **R — Repudiation:** Denying that an action was performed.
- **I — Information Disclosure:** Unauthorized access to sensitive data.
- **D — Denial of Service:** Making a system unavailable.
- **E — Elevation of Privilege:** Gaining unauthorized elevated permissions.

### Common Software Security Vulnerabilities

- **SQL Injection:** Malicious SQL code inserted into input fields.
- **Cross-Site Scripting (XSS):** Malicious scripts injected into web pages viewed by users.

- **Buffer overflow:** Writing beyond allocated memory boundaries.
- **Insecure deserialization:** Executing malicious code embedded in serialized objects.
- **Broken authentication:** Weak session management or credential storage.

### Security Assessment Standards

- **OWASP Top 10:** Most critical web application security risks — industry reference for web security.
- **NIST SP 800-115:** Technical guide to information security testing and assessment.
- **ISO/IEC 27001:** Information security management system standard.
- **Common Criteria (ISO 15408):** International standard for security evaluation of IT products.

---

## 15. Inspections and Reviews

*Inspections and reviews are formal and semi-formal static V&V techniques where a group of people systematically examine a software artifact (document, design, code) to detect defects without executing the software.*

### Types of Reviews

- **Walkthrough:** Author presents the work product to peers; informal, author-led.
- **Peer review (informal):** One or two colleagues review code or documents without formal process.
- **Technical review:** Structured review by peers to evaluate technical content against criteria.
- **Formal inspection:** Most rigorous — uses defined roles, checklists, exit criteria, and defect logging.

### Fagan Inspection Process (Formal Inspection)

- **Step 1 — Planning:** Moderator selects team, distributes materials, schedules meeting.
- **Step 2 — Overview:** Author explains the work product to the inspection team.
- **Step 3 — Preparation:** Each inspector individually reviews material, noting defects.
- **Step 4 — Inspection meeting:** Team reviews document together; reader paraphrases, inspectors raise defects.
- **Step 5 — Rework:** Author corrects all defects identified.
- **Step 6 — Follow-up:** Moderator verifies all defects have been corrected.

### Inspection Roles

- **Moderator:** Leads the inspection, ensures process compliance, collects metrics.
- **Author:** Created the work product under inspection; answers questions.
- **Reader:** Reads through the material aloud during the inspection meeting.
- **Inspector:** Independently prepares and raises defects during meeting.
- **Recorder/Scribe:** Logs all defects and actions identified.

### Reviews vs Inspections

| Aspect | Review | Inspection |
|---|---|---|
| Formality | Low to medium | High |
| Roles | Not predefined | Strictly defined |
| Checklists | Optional | Mandatory |

| | | |
|---|---|---|
| Metrics collected | Rarely | Always |
| Follow-up required | Optional | Mandatory |

### Defect Classification in Inspections

- **Major defect:** Would cause system failure or significant malfunction if not corrected.
- **Minor defect:** Deviation from standards that would not cause failure (typos, style violations).
- **Question:** Something the reviewer needs clarification on — not yet classified as defect.

---

## 16. Principles of Software Validation

*Software validation principles are the fundamental rules governing how validation activities are conducted to confirm that software satisfies user requirements and is fit for its intended use.*

### Core Validation Principles

- **Validation against requirements:** Every validation test must trace to a specified requirement.
- **Independence:** Validation is most effective when conducted by personnel not involved in development.
- **Completeness:** All requirements must be covered by validation activities.
- **Documentation:** All validation activities, results, and conclusions must be formally documented.
- **Reproducibility:** Validation must be conducted with sufficient documentation to allow repetition.
- **Risk-based approach:** Allocate validation effort proportional to the criticality of requirements.

### Validation vs. Verification — Principles

- Validation cannot substitute for verification — both must be performed.
- Passing all verification checks does not guarantee validation success.
- A system can be verified (built correctly per spec) but fail validation (spec was wrong).

### Validation Evidence

- Test execution records with actual vs expected results.
- Traceability matrix linking each requirement to validating test(s).
- Formal acceptance test report signed by authorized stakeholder.
- Discrepancy reports documenting any failures and resolutions.

### FDA 21 CFR Part 11 — Regulated Validation

- Applies to pharmaceutical and medical device software.
- Requires validated systems for electronic records and signatures.
- Mandates Installation Qualification (IQ), Operational Qualification (OQ), Performance Qualification (PQ).
  - IQ — System installed correctly per specifications.
  - OQ — System operates correctly within defined ranges.
  - PQ — System performs correctly under actual use conditions.

---

## 17. Software Verification

*Software verification is the process of evaluating software artifacts at each development phase to ensure they satisfy the conditions imposed at the start of that phase, without executing the software.*

### Verification Activities by Phase

- **Requirements verification:**
  - Check completeness — are all user needs captured?
  - Check consistency — do requirements contradict each other?
  - Check feasibility — can requirements be implemented within constraints?
  - Check testability — can each requirement be verified by a test?
- **Design verification:**
  - Confirm design addresses all requirements.
  - Check internal consistency of design components.
  - Verify design follows architectural standards.
- **Code verification:**
  - Code reviews against design and coding standards.
  - Static analysis — type checking, null pointer detection, unreachable code.
  - Unit test traceability to design specifications.

### Static Verification Techniques

- **Code reviews:** Line-by-line examination of code by developers.
- **Static analysis tools:** Automated analysis of code without execution — SonarQube, Coverity, Lint.
- **Model checking:** Formal technique exploring all system states to find violations of properties.
- **Abstract interpretation:** Over-approximates all possible program behaviors to prove properties.

### Formal Verification

- Proves mathematical correctness of software against formal specification.
- Techniques: theorem proving (Coq, Isabelle), model checking (SPIN, NuSMV).
- Computationally expensive — applied to small, critical components.
- Guarantees absence of certain classes of defects (not all defects).

### Verification Completeness

- Verification is complete when all phase exit criteria are satisfied.
- Exit criteria typically include: all defects resolved, all checklists complete, metrics within targets.
- Sign-off by SQA and management documents completion of each verification phase.

---

## 18. Planning for Software Quality Assurance

---

*SQA planning is the process of defining which SQA tasks will be performed, by whom, when, and with what resources, resulting in a formal Software Quality Assurance Plan (SQAP).*

### IEEE 730 — SQAP Structure

- **Section 1 — Purpose:** State the purpose of the SQAP and the software it covers.
- **Section 2 — Reference documents:** List standards, procedures, and contracts referenced.
- **Section 3 — Management:** Organizational structure, tasks, and responsibilities.
- **Section 4 — Documentation:** List all required documents (SRS, SDD, test plans, etc.).
- **Section 5 — Standards, practices, conventions:** Standards to be applied.

- **Section 6 — Reviews and audits:** Types, timing, and entry/exit criteria.
- **Section 7 — Test:** Reference to software test plan.
- **Section 8 — Problem reporting and corrective action:** Process for documenting and resolving issues.
- **Section 9 — Tools, techniques, and methodologies:** SQA tools to be used.
- **Section 10 — Media control:** Procedures for controlling software media.
- **Section 11 — Supplier control:** How external supplier quality is managed.
- **Section 12 — Records collection, maintenance, retention:** How SQA records are managed.
- **Section 13 — Training:** SQA training requirements.
- **Section 14 — Risk management:** Identification and mitigation of quality risks.

### SQA Planning Considerations

- Tailor the plan to project size, criticality, and contractual requirements.
- Establish realistic schedules that integrate SQA activities into the development schedule.
- Define clear entry and exit criteria for every review and audit.
- Ensure all team members understand and accept their SQA responsibilities.

---

## 19. Software Quality Assurance (SQA) Plans

*An SQA Plan (SQAP) is a formal document that specifies the SQA activities, resources, schedules, and standards to be applied throughout a software project.*

### Key Characteristics of an Effective SQAP

- Specific and measurable — contains quantitative quality targets, not vague goals.
- Comprehensive — covers all phases of the SDLC.
- Realistic — schedules and resources are achievable.
- Approved — formally reviewed and approved by management before project begins.
- Maintained — updated as the project evolves.

### SQAP vs Other Plans

| Plan | Focus | Contains |
|---|---|---|
| SQAP | Quality assurance process | Reviews, audits, standards, metrics |
| Test Plan | Testing activities | Test strategy, test cases, schedule |
| V&V Plan (SVVP) | Verification & validation activities | V&V tasks per SDLC phase |
| Configuration Mgmt Plan | Change control | Version control, baseline procedures |

### Common SQAP Deficiencies

- Too generic — lacks project-specific details.
- Not integrated into the project schedule — treated as a separate parallel activity.
- No measurable quality targets — qualitative goals only.
- Not updated after initial creation — becomes obsolete quickly.
- Not enforced — exists as documentation but not followed in practice.

---

# 20. SQA — Organizational Level Initiatives

*Organizational-level SQA initiatives are enterprise-wide programs that standardize quality practices across all projects, rather than within a single project.*

### Organizational SQA Structures

- **Centralized SQA:** A dedicated SQA department serves all projects in the organization.
  - Pros: Consistent standards, specialized expertise, economies of scale.
  - Cons: May be disconnected from project realities, slower response.
- **Distributed SQA:** SQA engineers are embedded within individual project teams.
  - Pros: Better project knowledge, faster feedback.
  - Cons: Inconsistent standards across projects, lack of independence.
- **Hybrid model:** Central SQA defines standards; project-level SQA implements them.

### Organizational SQA Initiatives

- **Process improvement programs:** Structured efforts to systematically improve development processes (CMMI appraisals, Six Sigma projects).
- **Metrics programs:** Organization-wide collection and analysis of quality metrics across all projects.
- **Standards programs:** Development and maintenance of organizational coding, design, and process standards.
- **Training programs:** Regular training in SQA techniques, tools, and standards for all staff.
- **Lessons learned repositories:** Organizational databases capturing defect patterns and process improvements from past projects.

### Process Asset Library (PAL)

- Repository of reusable process artifacts — templates, standards, checklists, metrics baselines.
- Used across projects to ensure consistency and reduce rework.
- Core component of CMMI Level 3 organizations.
- Must be actively maintained and updated based on project experiences.

---

# 21. SQA Planning — Observations, Numbers, Results

*SQA planning effectiveness is measured through systematic collection of quantitative observations (metrics) that produce numbers which are analyzed to yield actionable results for process improvement.*

### Observations in SQA

- Defects found per review, inspection, or test phase.
- Number of pages/KLOC reviewed per hour (review rate).
- Number of rework hours required per defect found.
- Number of test cases executed vs planned.
- Frequency of process deviations found in audits.

### Numbers — Key SQA Metrics

- **Defect density:** Defects / KLOC — measures product quality.
- **Defect removal efficiency (DRE):** (Defects removed before release) / (Total defects) × 100%.
  - Industry benchmark: DRE > 95% indicates mature SQA process.

- **Phase containment effectiveness:** % of defects found in the phase where they were introduced.
- **Review efficiency:** Major defects found per review hour.
- **Test coverage:** % of requirements or code covered by tests.

### Results — Using SQA Data

- Compare actual metrics against planned targets to identify process deviations.
- Track trends over time — improving or degrading defect rates indicate process health.
- Identify high-defect modules for targeted improvement or redesign.
- Use DRE to determine readiness for release.
- Report results to management to support go/no-go decisions.

### Goal-Question-Metric (GQM) Approach

- **Goal:** Define a quality objective (e.g., reduce post-release defect rate by 20%).
- **Question:** What questions must be answered to know if the goal is met?
- **Metric:** What specific measurements answer each question?
- GQM ensures every metric collected has a defined purpose tied to a goal.

---

## 22. Software Testing

*Software testing is the process of executing a program or system with the intent of finding defects, and evaluating whether it meets specified requirements.*

### Testing Levels

- **Unit testing:** Testing individual components (functions, classes) in isolation.
- **Integration testing:** Testing how components interact when combined.
- **System testing:** Testing the complete, integrated system against system requirements.
- **Acceptance testing:** Validation by end users or client that system meets business requirements.

### Testing Types

- **Functional testing:** Verifying each function of the software produces expected output.
- **Non-functional testing:** Performance, load, stress, security, usability testing.
- **Regression testing:** Re-testing after changes to ensure existing functionality is unaffected.
- **Smoke testing:** Quick preliminary test to verify basic functionality before deeper testing.
- **Exploratory testing:** Simultaneous learning, test design, and execution without predefined test cases.

### Test Case Components

- **Test case ID:** Unique identifier.
- **Preconditions:** State the system must be in before the test.
- **Test inputs:** Specific data values to be entered.
- **Expected result:** What the system should produce.
- **Actual result:** What the system actually produced (recorded during execution).
- **Pass/Fail:** Comparison of expected vs actual result.

### Testing Principles (Dijkstra / ISTQB)

- Testing shows presence of defects — it cannot prove their absence.

- Exhaustive testing is impossible — use risk-based selection.
- Test early — defect cost increases with each phase passed.
- Defects cluster — a small number of modules contain most defects (Pareto principle).
- Pesticide paradox — repeating the same tests stops finding new defects; tests must evolve.
- Testing is context-dependent — safety-critical software requires more rigorous testing.
- Absence-of-errors fallacy — a bug-free system that doesn't meet user needs is not quality.

---

# 23. Specification-Based Test Construction Techniques

*Specification-based (black-box) test construction techniques derive test cases from the functional specification or requirements of the software, without knowledge of its internal structure.*

### 1. Equivalence Partitioning

- Divides the input domain into classes (partitions) where all values in a class are expected to behave the same.
- **Valid partition:** Input values that the system should accept.
- **Invalid partition:** Input values that the system should reject.
- Select one representative value from each partition.
- **Example:** For age field (1–120): valid partition [1–120], invalid partitions [<1] and [>120].

### 2. Boundary Value Analysis (BVA)

- Tests values at the exact boundaries of equivalence partitions.
- Defects cluster at boundaries — most errors occur at edge cases.
- For a range [min, max], test: min-1, min, min+1, max-1, max, max+1.
- **Example:** Age field [1–120]: test values 0, 1, 2, 119, 120, 121.

### 3. Decision Table Testing

- Models all combinations of input conditions and their corresponding actions.
- Each column represents a test case (unique combination of conditions).
- Ensures that all business rules and condition combinations are covered.
- Most useful for systems with complex conditional logic.

### 4. State Transition Testing

- Derives test cases from state machine model of system behavior.
- Test cases cover transitions between states triggered by events.
- Coverage levels: all states visited, all transitions exercised, all invalid transitions tested.
- **Example:** ATM states — Idle, Card Inserted, PIN Entered, Transaction, Card Ejected.

### 5. Use Case Testing

- Derives test scenarios from use cases defined in requirements.
- Covers main success scenarios and all alternative/exception flows.
- Ensures system behavior matches documented user interactions.

### Technique Selection Guide

| Technique | Best For | Coverage Type |
|---|---|---|
| Equivalence Partitioning | Input validation | Input domain coverage |
| BVA | Numeric ranges, limits | Boundary coverage |
| Decision Table | Complex business logic | Condition combination coverage |
| State Transition | Event-driven systems | State/transition coverage |
| Use Case Testing | End-to-end scenarios | Requirement coverage |

---

# 24. White-Box and Grey-Box Testing

*White-box testing derives test cases from the internal structure (source code, design) of the software; grey-box testing uses partial knowledge of the internals to supplement black-box techniques.*

## White-Box Testing

- Also called: glass-box, structural, or code-based testing.
- Requires access to the source code.
- Primarily performed by developers during unit testing.

### *White-Box Coverage Criteria*

- **Statement coverage:** Every statement in the code is executed at least once.
  - Minimum acceptable criterion — does not guarantee branch coverage.
- **Branch coverage (Decision coverage):** Every branch of every decision is executed (true and false).
  - Subsumes statement coverage.
- **Condition coverage:** Each individual condition in a decision is evaluated to true and false.
- **MC/DC (Modified Condition/Decision Coverage):** Each condition independently affects outcome.
  - Required for DO-178C Level A (aviation safety-critical software).
- **Path coverage:** Every possible path through the code is executed.
  - Infeasible in practice for large programs — exponential number of paths.

## Grey-Box Testing

- Tester has partial knowledge of internal workings — often has access to database schemas, algorithms, or architecture.
- Combines black-box test design with white-box verification.
- Common in integration testing, API testing, and security testing.
- Advantages: More targeted than pure black-box; less overhead than full white-box.

## Comparison

| Aspect | Black-Box | Grey-Box | White-Box |
|---|---|---|---|
| Code access | None | Partial | Full |
| Perspective | User/spec | Informed user | Developer |
| Defects found | Functional | Integration/logic | Code-level |
| Coverage measured | Requirements | API/data paths | Code paths |

---

# 25. Comprehensive Software Testing Techniques for SDLC

*Comprehensive software testing spans multiple testing types applied systematically across SDLC phases to ensure complete coverage of functional, non-functional, and structural requirements.*

### Testing Across SDLC Phases

| SDLC Phase | Testing Activity | Technique |
|---|---|---|
| Requirements | Requirements review testing | Inspections, walkthroughs |
| Design | Design review | Static analysis, FTA |
| Coding | Unit testing, code review | White-box, static analysis |
| Integration | Integration testing | Big bang, incremental |
| System | System testing | Black-box, performance |
| Acceptance | UAT, beta testing | Use case testing, exploratory |

### Integration Testing Strategies

- **Big Bang:** All components integrated simultaneously and tested.
  - Pros: Simple. Cons: Hard to isolate defects.
- **Top-down:** Higher-level modules integrated first; stubs replace lower modules.
  - Tests interfaces early; stubs require extra effort.
- **Bottom-up:** Lower-level modules integrated first; drivers replace higher modules.
  - Natural for reusable component development.
- **Sandwich (Hybrid):** Combines top-down and bottom-up concurrently.

### Non-Functional Testing Types

- **Performance testing:** Measures response time, throughput under expected load.
- **Load testing:** Tests behavior under peak expected load.
- **Stress testing:** Tests beyond normal operating capacity to find breaking point.
- **Usability testing:** Evaluates ease of use with real users.
- **Compatibility testing:** Verifies system works across browsers, OS, devices.
- **Security testing:** Identifies vulnerabilities and exposure to attacks.
- **Recovery testing:** Tests ability to recover from failures.

---

# 26. Control Flow Oriented Test Construction Techniques

*Control flow oriented testing uses the control flow graph (CFG) of a program to derive test cases that execute paths, branches, or conditions in the code.*

### Control Flow Graph (CFG)

- A CFG represents the flow of control through a program.
- **Nodes:** Represent statements or blocks of sequential statements.
- **Edges:** Represent possible transfers of control (branches, loops, gotos).
- **Entry node:** Start of the program/function.
- **Exit node:** End of the program/function.

### Coverage Criteria Based on CFG

- **Node coverage (Statement coverage):** Every node (statement) in the CFG is executed.
  - Weakness: Can miss untested branches.
- **Edge coverage (Branch coverage):** Every edge (branch) in the CFG is traversed.
  - Stronger than node coverage.
- **Path coverage:** Every distinct path from entry to exit is executed.
  - Usually infeasible — exponential paths in loops.
- **Prime path coverage:** Cover all maximal simple paths (no repeated nodes except possibly first=last).
  - Practical compromise between branch and full path coverage.

### Cyclomatic Complexity

- Quantifies the number of independent paths in a CFG.
- **Formula:** V(G) = E – N + 2P, where E = edges, N = nodes, P = connected components.
- **Simplified:** V(G) = number of predicate nodes + 1.
- V(G) gives the minimum number of test cases needed for branch coverage.
- V(G) > 10 indicates high complexity — candidate for refactoring.

### Basis Path Testing

- Technique using cyclomatic complexity to derive a basis set of independent paths.
- **Step 1:** Draw the CFG from the code.
- **Step 2:** Calculate V(G).
- **Step 3:** Identify V(G) independent paths.
- **Step 4:** Design one test case per independent path.
- Guarantees 100% branch coverage with minimum test cases.

---

## 27. Data Flow Oriented Test Construction Techniques

*Data flow oriented testing focuses on the definition and use of variables within a program to derive test cases that exercise sequences from where a variable is defined (written) to where it is used (read).*

### Core Concepts

- **Definition (def):** A location in code where a variable is assigned a value.
- **Use (use):** A location where a variable's value is accessed.
  - **c-use (computational use):** Variable used in a computation.
  - **p-use (predicate use):** Variable used in a condition/predicate.
- **def-use pair:** A pair (d, u) where d is a definition and u is a use of the same variable with a definition-clear path from d to u.
- **Definition-clear path:** A path from def to use where the variable is not redefined.

### Data Flow Coverage Criteria

- **All-defs:** At least one use is covered for each definition. (Weakest)
- **All-uses:** Every def-use pair is covered by at least one test path.
- **All-p-uses:** All predicate uses for each definition are covered.
- **All-c-uses:** All computational uses for each definition are covered.
- **All-du-paths:** All definition-use paths are covered. (Strongest)

### Data Flow Anomalies

- **Defined but never used (du):** Variable defined but never referenced — potential dead code.
- **Used before defined (ur):** Variable used without being initialized — undefined behavior.
- **Defined twice without use (dd):** Variable redefined before its first definition is used — wasted computation.

### Data Flow vs Control Flow Testing

| Aspect | Control Flow | Data Flow |
|---|---|---|
| Focus | Execution paths (branches, loops) | Variable def-use pairs |
| Graph type | Control Flow Graph | Def-Use Graph |
| Defects found | Missing branches, dead code | Uninitialized variables, incorrect values |
| Complexity | Lower | Higher — more pairs to cover |

---

## 28. Clean-Room Approach to Quality Assurance

*The Cleanroom approach is a software development methodology that applies statistical quality control and formal methods to produce software with measurable, certified reliability prior to first execution.*

### Core Principles of Cleanroom

- **Defect prevention over defect removal:** Prevent defects through rigorous specification and incremental development.
- **Statistical quality control:** Test using statistically designed test cases drawn from the usage profile.
- **No unit testing by developers:** Developers do not execute and debug their own code.
- **Correctness verification:** Teams use formal mathematical reasoning to verify correctness before integration.
- **Incremental development:** Software is developed and certified in increments, each of which passes reliability certification.

### Cleanroom Process Steps

- **Step 1 — Box-structured specification:** System defined using Black Box (external behavior), State Box (state data), and Clear Box (internal logic).
- **Step 2 — Incremental development:** Each increment implements a portion of the Black Box specification.
- **Step 3 — Correctness verification:** Development team verifies code correctness using formal proofs (no test execution).
- **Step 4 — Statistical testing:** Independent test team creates test cases based on operational profile.
- **Step 5 — Reliability certification:** Mean Time To Failure (MTTF) is measured and certified against target.

### Box Structures

- **Black Box:** Defines what the system does — external behavior, inputs, outputs, state transitions.
- **State Box:** Defines the data state retained between uses of the system.
- **Clear Box:** Defines how the system implements the State Box specification — internal logic.

### Cleanroom vs Traditional Development

| Aspect | Traditional | Cleanroom |
|---|---|---|

| | | |
|---|---|---|
| Unit testing | Developers test own code | No execution until certification |
| Defect removal | Debug and retest | Formal correctness verification |
| Test design | Ad hoc or coverage-based | Statistically designed from usage profile |
| Reliability measurement | Rarely done formally | Certified MTTF per increment |

---

# 29. Product Quality and Process Quality

*Product quality refers to the degree to which a software product possesses desirable characteristics; process quality refers to how well the development processes are defined, followed, and optimized.*

### Product Quality — ISO/IEC 25010 Characteristics

- **Functional suitability:** Functions present, correct, and appropriate.
- **Performance efficiency:** Response time, throughput, resource utilization.
- **Compatibility:** Coexistence and interoperability with other systems.
- **Usability:** Ease of learning, use, accessibility.
- **Reliability:** Maturity, availability, fault tolerance, recoverability.
- **Security:** Confidentiality, integrity, non-repudiation, authenticity.
- **Maintainability:** Modularity, reusability, analyzability, modifiability, testability.
- **Portability:** Adaptability, installability, replaceability.

### Process Quality

- A high-quality process tends to produce high-quality products — but is not a guarantee.
- Measured by: defect introduction rate, rework effort, schedule adherence, process compliance.
- Improved through: CMMI assessments, Six Sigma projects, lessons learned programs.

### Relationship Between Process and Product Quality

- A poor process can still produce acceptable products by chance — not sustainably.
- A good process consistently produces products meeting quality targets.
- Improving process quality reduces variability in product quality.
- Neither alone is sufficient — both product and process must be measured and controlled.

### Product Quality Measurement

- **Internal quality measures:** Code complexity, coupling, cohesion, code duplication.
- **External quality measures:** Defect density in field, customer-reported defects, uptime.
- **In-use quality measures:** User satisfaction scores, task completion rates, error frequency.

---

# 30. Standards for Process Quality and Product Quality

*Process quality standards define how software development processes should be structured and executed; product quality standards define the characteristics and attributes software products must possess.*

### Process Quality Standards

- **ISO 9001:2015:** Requirements for a quality management system — document processes, measure performance, continually improve.
- **CMMI for Development (CMMI-DEV):** Framework for process improvement with 5 maturity levels.
- **ISO/IEC 12207:** Software lifecycle processes — defines activities for acquisition, supply, development, operation, maintenance.
- **ISO/IEC 15504 (SPICE):** Software process assessment framework — capability levels 0–5.
- **IEEE 730:** SQA plan standard — specifies SQAP content and structure.

### Product Quality Standards

- **ISO/IEC 25010 (SQuaRE):** Software product quality model — 8 quality characteristics with sub-characteristics.
- **ISO/IEC 25040:** Evaluation process for software product quality.
- **IEEE 1061:** Software quality metrics methodology.
- **DO-178C:** Aviation software product quality and development assurance.
- **IEC 62304:** Medical device software lifecycle and quality requirements.

### Standards Adoption Process

- **Step 1 — Selection:** Identify applicable standards based on domain, contracts, regulations.
- **Step 2 — Tailoring:** Adapt standard requirements to organizational and project context.
- **Step 3 — Implementation:** Integrate standard requirements into processes and procedures.
- **Step 4 — Compliance verification:** Audit and verify that standards are being followed.
- **Step 5 — Maintenance:** Track and adopt new standard versions.

---

## 31. Walkthroughs and Inspections

*Walkthroughs and inspections are structured peer review techniques in which team members examine software artifacts to identify defects, with inspections being more formal and rigorous than walkthroughs.*

### Walkthrough

- Author guides reviewers through the work product to explain it and gather feedback.
- Goal: Understanding, defect detection, knowledge sharing.
- Informal — no formal defect logging required.
- Can be done at any stage — requirements, design, code, test plans.

#### *Walkthrough Process*

- **Step 1 — Preparation:** Author distributes materials; reviewers prepare individually.
- **Step 2 — Presentation:** Author walks through document step by step.
- **Step 3 — Discussion:** Reviewers raise issues; scribe records.
- **Step 4 — Follow-up:** Author corrects issues (no formal verification required).

### Inspection (Fagan Inspection)

- Most formal review type — predefined roles, checklists, metrics, follow-up.
- Goal: Defect detection and removal — not design discussion.
- Proven to remove 60–90% of defects before testing.

#### *Inspection Process (6 Steps)*

- **1. Planning:** Moderator forms team, distributes work product.
- **2. Overview:** Author briefs team (optional for experienced teams).
- **3. Preparation:** Each inspector individually reviews document with checklists.
- **4. Meeting:** Moderator leads; reader paraphrases; inspectors raise defects.
- **5. Rework:** Author corrects all defects.
- **6. Follow-up:** Moderator verifies all defects addressed; may re-inspect if major.

### Walkthrough vs Inspection

| Aspect | Walkthrough | Inspection |
|---|---|---|
| Formality | Low | High |
| Leader | Author | Moderator (not author) |
| Preparation | Optional | Mandatory — individual prep |
| Checklists | No | Yes |
| Metrics | No | Yes — defect rates collected |
| Follow-up | Informal | Mandatory — verified by moderator |
| Effectiveness | Moderate | High — industry benchmark 60–90% DRE |

---

## 32. Structure, Checklist, and Audits

*Structured reviews use defined procedures and documentation; checklists ensure completeness of reviews; audits are independent examinations to verify compliance with standards and procedures.*

### Structured Reviews

- Reviews must have: defined entry criteria, defined process steps, defined exit criteria.
- Work product must be distributed sufficiently in advance for preparation.
- All defects are logged, classified, and tracked to closure.
- Review results are formally recorded and archived.

### Checklists

- A checklist is a predefined list of items to verify during a review or inspection.
- **Requirements checklists:** Are requirements complete? Consistent? Testable? Unambiguous?
- **Design checklists:** Is coupling minimized? Are interfaces defined? Is error handling addressed?
- **Code checklists:** Are variables initialized? Are loop terminations correct? Are exceptions handled?
- **Test plan checklists:** Are all requirements covered? Are test environments specified?

#### *Checklist Best Practices*

- Keep checklists concise — 10–20 items maximum per phase.
- Update checklists based on defects found in previous projects.
- Customize for project type, language, and domain.
- Treat checklist items as binary — yes/no, not subjective ratings.

### Audits

- An audit is an independent, documented examination of a project to determine compliance with plans, standards, and procedures.

***Types of Audits***

- **Process audit:** Verifies that defined processes are being followed.
- **Product audit:** Examines a specific work product for conformance to standards.
- **Functional audit:** Verifies all functional requirements are met before delivery.
- **Physical audit:** Verifies that all specified documentation and components exist and are consistent.
- **Configuration audit:** Verifies consistency between software documentation and code at a baseline.

***Audit Process***

- **Step 1 — Initiation:** Audit scope, objectives, team, and schedule are defined.
- **Step 2 — Preparation:** Auditors review relevant standards and prior audit reports.
- **Step 3 — Execution:** Auditors examine evidence, interview personnel, observe processes.
- **Step 4 — Report:** Findings are documented — nonconformances, observations, recommendations.
- **Step 5 — Follow-up:** Corrective actions are tracked until closure.

---

# 33. Roles and Responsibilities in Reviews and Inspections

*Defined roles and responsibilities in reviews and inspections ensure that all necessary activities are performed systematically and that no single person acts as both author and evaluator of their own work.*

## Core Roles

- **Moderator (Inspection Leader):**
  - Plans, leads, and controls the inspection process.
  - Selects inspection team members.
  - Ensures meeting stays focused on defect detection, not problem solving.
  - Collects and reports inspection metrics.
  - Conducts follow-up to verify defect correction.
- **Author:**
  - Created the work product under review.
  - Provides clarifications during review — does not defend or argue.
  - Responsible for correcting all identified defects.
- **Reader:**
  - Presents the work product during inspection meeting (paraphrases, does not read verbatim).
  - Controls the pace of the meeting.
- **Inspector / Reviewer:**
  - Reviews material individually during preparation phase.
  - Records and raises defects during the meeting.
  - Should bring prepared defect list — not discover defects only during meeting.
- **Recorder / Scribe:**
  - Documents all defects identified during the meeting.
  - Records defect type, location, severity, and resolution.

## Responsibilities Matrix

| Activity | Moderator | Author | Reader | Inspector | Recorder |
|---|---|---|---|---|---|
| Plan inspection | ✓ | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Distribute materials | ✓ | ✓ | | | |
| Individual prep | | ✓ | ✓ | ✓ | |
| Lead meeting | ✓ | | | | |
| Paraphrase product | | | ✓ | | |
| Raise defects | | | | ✓ | |
| Log defects | | | | | ✓ |
| Rework defects | | ✓ | | | |
| Verify rework | ✓ | | | | |

### Independence in Reviews

- Author should not moderate their own work — independence is essential.
- For highest-criticality systems, reviewers should be from outside the development team.
- Management should not participate in technical reviews — it inhibits open discussion.

---

## 34. How to Make Reviews and Inspections Most Effective

*Review and inspection effectiveness is maximized through adequate preparation, disciplined meeting management, proper team composition, systematic defect logging, and continuous improvement based on collected metrics.*

### Before the Meeting — Preparation

- Distribute materials at least 2–3 days in advance.
- Establish entry criteria: work product must be complete enough to inspect.
- Each inspector independently reviews material and prepares a defect list before meeting.
- Meeting should not begin if any inspector has not prepared.
- Optimal inspection rate: 100–200 lines of code per hour; 8–10 pages of documentation per hour.

### During the Meeting

- Limit meeting duration to 2 hours maximum — effectiveness drops sharply after.
- Optimal team size: 3–6 people. Larger teams have diminishing returns.
- Focus only on defect identification — not solving problems during meeting.
- Moderator enforces discipline: no side discussions, no blame.
- All defects are logged with precise location, type, and severity.

### After the Meeting

- Author corrects all logged defects within agreed timeframe.
- Moderator performs follow-up review to verify corrections.
- Major defects may require re-inspection of corrected sections.
- Metrics are recorded: pages inspected, defects found, preparation hours, meeting hours.

### Key Success Factors

- **Management support:** Management must schedule time for reviews and not treat them as optional.

- **No blame culture:** Defects are properties of the product, not evidence of developer incompetence.
- **Consistent process:** Use the same process every time — ad hoc reviews are far less effective.
- **Updated checklists:** Regularly revise checklists based on defects that were missed.
- **Metrics feedback loop:** Use review data to identify problem areas and improve development practices.

### Common Effectiveness Killers

- Insufficient preparation time — inspectors discover defects only during meeting.
- Mixing defect identification with problem solving in the same meeting.
- Management attendance — developers self-censor to avoid embarrassment.
- No follow-up — defects are logged but corrections are never verified.
- Inspecting too large a work product in one session — exceeds cognitive capacity.

### Metrics for Review Effectiveness

- **Defect detection rate:** Defects per page or per KLOC found during review.
- **Inspection rate:** Pages or LOC reviewed per hour — if too fast, quality suffers.
- **Preparation effort:** Hours spent individually preparing vs meeting duration.
- **Rework rate:** Hours of rework per defect found — indicates defect severity distribution.
- **Phase containment:** % of defects from this phase caught in this review vs escaping to later phases.

---

— End of Software Verification & Validation Notes —